Étape 9 : fourmis et fourmilières, partie 3.

Jamila Sam & Jean-Cédric Chappelier, 2018

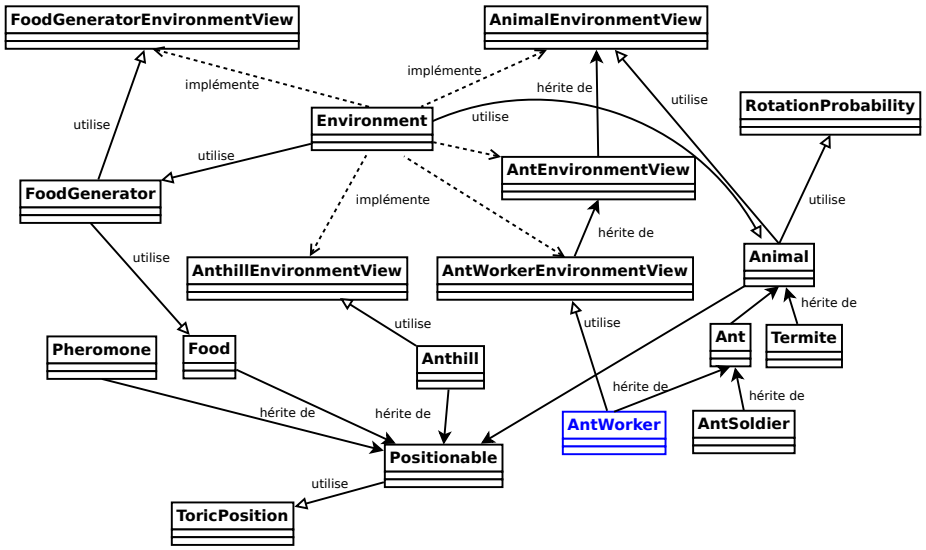
Version : 1.0

But

Implémenter un comportement propre aux fourmis ouvrières.

Description

Jusqu'ici, nous avons mis en place une conception qui, grâce à usage judicieux de l'héritage, du polymorphisme et de l'encapsulation au travers d'interfaces, permet de mettre en œuvre les comportements spécifiques des différents animaux de la simulation, sans pour autant leur donner trop de liberté quant à leur impact possible sur l'environnement.



Nous allons maintenant commencer à tirer réellement parti de cette conception et introduire un comportement spécifique aux ouvrières : leur capacité à chercher de la nourriture et en ramener à leur fourmilière.

Il n'y aura donc pas de nouveaux composants introduits lors de cette étape, mais uniquement des retouches à la classe `AntWorker` comme le montre la figure 1.

Implémentation

La méthode `seekForFood()`

Dans la classe `AntWorker`, nous avons précédemment introduit la méthode

```
void seekForFood(AntWorkerEnvironmentView env, Time dt)
```

qui se contente pour l'heure d'invoquer la méthode de déplacement.

Il vous est demandé de la compléter afin qu'elle régisse le comportement spécifique à une ouvrière :

- si elle ne transporte rien (aucune nourriture) : lorsqu'elle rencontre une source de nourriture, elle en puise (méthode `takeQuantity()` de `Food`) une partie (paramètre `ANT_MAX_FOOD` de type `double`), puis fait demi-tour ;
- **puis** (notez donc que ceci peut se produire *juste après* avoir ramassé de la nourriture) si elle transporte de la nourriture , elle essaye de la déposer (méthode `dropFood()` de l'environnement) et si elle y parvient : remet sa quantité de nourriture à 0 et fait demi-tour.

Notes :

- la fourmi ouvrière ne peut pas prélever plus que la quantité qu'elle peut porter, donnée par `ANT_MAX_FOOD` ;
- utiliser les méthodes `dropFood()` et `getClosestFoodForAnt()` de l'interface `AntWorkerEnvironmentView` que vous avez écrites précédemment ;
- pour faire faire un tour complet sur elle-même à la fourmi, il suffit d'ajouter `Math.PI` à son angle de direction ; si après cet ajout l'angle est plus grand que deux fois `Math.PI`, lui soustraire deux fois `Math.PI`.

Tests et soumission

Tests locaux

Pour une fois, nous ne pensons pas possible dans cette étape de faire des tests ponctuels dans le fichier `ch/epfl/moocprog/tests/Main.java` sans casser l'encapsulation et vous conseillons donc ici de n'utiliser **que** l'interface graphique pour tester globalement le comportement de tout le système. Créez pour cela des fichiers de paramétrages facilitant la visualisation de certains comportements.

Par exemple, vous pouvez créer dans le dossier `res` un fichier `config-step9.cfg` dont le contenu serait simplement le suivant:

```
Anthill:450, 350
```

Modifiez le contexte d'exécution de l'application graphique de sorte à utiliser plutôt ce fichier. Vous aurez donc dans le fichier `app/Context.java` plutôt ceci :

```
public final class Context {
    private static Application THE_APP;
    public static String CONFIG_PATH= "res/app.cfg";
    public static String INIT_PATH="res/config-step9.cfg";
```

Avoir une seule fourmilière sera en effet plus propice à l'observation des comportements nouvellement introduits (en tout cas un certain nombre).

Une fois l'application graphique lancée avec ce contexte, paramétrez là de sorte à n'avoir que des fourmis ouvrières (`ANTHILL_WORKER_PROB_DEFAULT` à 1). Vous pouvez aussi accélérer la création de tas de nourriture en mettant `FOOD_GENERATOR_DELAY` à 0.5 par exemple. En activant le bouton de debug `Quantity` vous devriez pouvoir vérifier que les ouvrières :

- font des va-et-vient entre des tas de nourriture et leur fourmilière: elles doivent être capable de prélever une quantité de nourriture (dont le tas se départit), la transporter, puis s'en délester dans leur fourmilière dont le stock s'accroît ;
- ignorent les tas de nourriture rencontrés lorsqu'elles sont chargées.

Une petite vidéo d'exemple d'exécution possible dans ce contexte est donné dans la matériel de la semaine 5.

Vous pouvez ensuite dans le même esprit, modifier le fichier `config-step9.cfg` pour y ajouter une seconde fourmilière par exemple :

```
Anthill:450, 350
Anthill:850, 750
```

et ainsi vérifier que les fourmis ne déposent de la nourriture que dans la fourmilière dont elles sont issues.

Soumission

Pour soumettre votre devoir au correcteur automatique, il faut créer un fichier ZIP contenant **tous** vos fichiers sources personnels, depuis la racine `ch/` ; ceux de cette étape, mais aussi ceux de l'étape précédente. Concrètement, pour ce devoir ci, votre fichier ZIP doit donc contenir exactement les fichiers suivants :

```
ch/epfl/moocprog/AnimalEnvironmentView.java
ch/epfl/moocprog/Animal.java
ch/epfl/moocprog/AntEnvironmentView.java
```

`ch/epfl/moocprog/AnthillEnvironmentView.java`
`ch/epfl/moocprog/Anthill.java`
`ch/epfl/moocprog/Ant.java`
`ch/epfl/moocprog/AntSoldier.java`
`ch/epfl/moocprog/AntWorkerEnvironmentView.java`
`ch/epfl/moocprog/AntWorker.java`
`ch/epfl/moocprog/Environment.java`
`ch/epfl/moocprog/FoodGeneratorEnvironmentView.java`
`ch/epfl/moocprog/FoodGenerator.java`
`ch/epfl/moocprog/Food.java`
`ch/epfl/moocprog/Pheromone.java`
`ch/epfl/moocprog/Positionable.java`
`ch/epfl/moocprog/RotationProbability.java`
`ch/epfl/moocprog/Termite.java`
`ch/epfl/moocprog/ToricPosition.java`

Note : si c'est plus pratique pour vous, vous *pouvez* aussi faire un zip qui contient tout le sous-répertoire `ch/epfl/moocprog`, y compris, donc, les fichiers que nous vous avons fournis. Ces fichiers seront simplement ignorés par le correcteur automatique (et remplacés par les nôtres). L'énoncé « *Procédure de soumission* » de la semaine 1 vous indique comment procéder. **Avant de passer à l'étape suivante, n'oubliez pas de faire une sauvegarde de l'étape en cours**, comme indiqué dans le même document.